TD numéro 1
Modularité, compilation séparée

Programmation impérative avancée, ENSIIE

Semestre 2, 2015–16

On rappelle l'interface d'un module pour les piles :

pile.h

```
typedef struct pile_base* pile;

pile vide();
bool est_vide(pile);
void push(pile*, int);
int pop(pile*);
```

On souhaite écrire un programme qui évalue des expressions arithmétiques écrites en notation polonaise inversée, comme par exemple `3 4 + 5 +` qui s'évalue en `12`. Pour cela, en plus du module pour les piles, on dispose des modules suivants :

Module `fichier`

- un type abstrait pour les flux ;
- une fonction `ouvre` qui prend une chaîne de caractère en argument et retourne un flux correspondant au fichier indiqué par la chaîne ;
- une fonction `ferme` qui prend un flux en argument et le ferme ;
- une fonction `lit_un_mot` qui prend un flux en argument et qui retourne une chaîne de caractère correspondant à un mot de l'entrée ;
- une fonction `fin_de_fichier` qui prend un flux en argument et qui retourne 1 ssi on a atteint la fin du flux.

Module `lexeme`

- un type abstrait pour les lexèmes, c'est-à-dire des éléments atomiques des expressions : constante entière ou opérateur plus ;
- une fonction `chaine_vers_lexeme` qui prend en argument une chaîne de caractère et qui retourne le lexème qui lui correspond ;
- une fonction `est_constante` qui prend un lexème en entrée et retourne 1 ssi c'est une constante entière ;
- une fonction `est_plus` qui prend un lexème en entrée et retourne 1 ssi c'est l'opérateur plus ;

— une fonction `val_constante` qui prend un lexème en entrée et retourne l'entier correspondant si c'est une constante (non défini sinon).

Module `evaluation`

— une fonction `eval` qui prend un flux en argument, évalue l'expression du flux en utilisant une pile et retourne l'entier correspondant à l'évaluation ;
— une fonction `main` qui appelle `eval` sur le fichier correspondant au premier argument passé en ligne de commande et affiche le résultat de l'évaluation sur la sortie standard.

1. Quelles sont les relations de dépendance entre les modules ?
2. Quelle commande faut-il écrire manuellement pour compiler le module `fichier` ?
3. En supposant que tous les modules sont compilés, quelle commande faut-il taper manuellement pour faire l'édition de liens ?
4. On modifie l'interface de `fichier`. Quel(s) module(s) faut-il recompiler ?
5. On modifie l'implémentation de `fichier`. Quel(s) module(s) faut-il recompiler ?
6. Écrire les interfaces des modules `fichier`, `lexeme` et `evaluation`.
7. Écrire l'implémentation du module `evaluation`.

   Rappel : pour accéder aux arguments de la ligne de commande en C, on définit la fonction `main` avec deux arguments `int argc, char **argv`. `argc` est le nombre d'arguments plus 1, `argv` est le tableau des arguments en tant que chaînes de caractères (le premier des éléments du tableau étant le nom du programme).
8. Écrire le `Makefile` correspondant au projet. On n'oubliera pas d'écrire une cible pour produire l'exécutable final qu'on appellera `prog`.
9. On souhaite rajouter la gestion de l'opérateur fois. Quels fichiers faut-il modifier a priori ? On distinguera interface et implémentation.